Channabasaveshwara Institute of Technology

(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)

(NAAC Accredited & ISO 9001:2015 Certified Institution)

NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.

.

BDA
LAB MANUAL

Name: Mrs. Mala K

Designation: ASSISTANT PROFESSOR

Department: ISE

Semester: VII Section: A

Subject & Subject code: BIG DATA ANALYTICS & BIS701

Channabasaveshwara Institute of Technology

(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)
((**NAAC Accredited & ISO 9001:2015 Certified Institution)**
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

SYLLABUS

BIG DATA ANALYTICS LAB

Sub Code: BIS701 **CIE Marks: 25**
Hrs/week: 02 **IPCC**

LIST OF EXPERIMENTS

At the end of the course, the student will be able to:

1. Identify and list various Big Data concepts, tools and applications.
2. Develop programs using HADOOP framework.
3. Make use of Hadoop Cluster to deploy Map Reduce jobs, PIG, HIVE and Spark programs.
4. Analyze the given data set and identify deep insights from the data set.
5. Demonstrate Text, Web Content and Link Analytics.

Sl. No.	*List of problems for which student should develop program and execute in the Laboratory*
1.	Install Hadoop and Implement the following file management tasks in Hadoop: Adding files and directories Retrieving files Deleting files and directories. Hint: A typical Hadoop workflow creates data files (such as log files) elsewhere and copies them into HDFS using one of the above command line utilities.
2.	Develop a MapReduce program to implement Matrix Multiplication
3.	Develop a Map Reduce program that mines weather data and displays appropriate messages indicating the weather conditions of the day.
4.	Develop a MapReduce program to find the tags associated with each movie by analyzing movie lens data.
5.	Implement Functions: Count – Sort – Limit – Skip – Aggregate using MongoDB
6.	Develop Pig Latin scripts to sort, group, join, project, and filter the data.
7.	Use Hive to create, alter, and drop databases, tables, views, functions, and indexes.
8.	Implement a word count program in Hadoop and Spark.
9.	Use CDH (Cloudera Distribution for Hadoop) and HUE (Hadoop User Interface) to analyze data and generate reports for sample datasets

Experiment 1: Hadoop Installation and File Management

Objective

Install Hadoop and implement file management tasks including adding, retrieving, and deleting files/directories.

Prerequisites

- Linux/Ubuntu OS (Recommended: Ubuntu 20.04 or later)
- Java JDK 8 or 11
- Minimum 4GB RAM

Step 1: Install Java

bash

```
# Update system packages
sudo apt update

# Install OpenJDK
sudo apt install openjdk-8-jdk -y

# Verify installation
java -version
javac -version

# Set JAVA_HOME
echo "export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64" >> ~/.bashrc
echo "export PATH=\$PATH:\$JAVA_HOME/bin" >> ~/.bashrc
source ~/.bashrc
```

Step 2: Create Hadoop User

bash

```
sudo adduser hadoop
sudo usermod -aG sudo hadoop
su - hadoop
```

Step 3: Install SSH and Configure Passwordless SSH

bash

```
sudo apt install openssh-server openssh-client -y
ssh-keygen -t rsa -P ""
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
chmod 640 ~/.ssh/authorized_keys
ssh localhost
```

Step 4: Download and Install Hadoop

bash

```
# Download Hadoop 3.3.6
wget https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz

# Extract
tar -xzf hadoop-3.3.6.tar.gz
sudo mv hadoop-3.3.6 /usr/local/hadoop
sudo chown -R hadoop:hadoop /usr/local/hadoop
```

Step 5: Configure Environment Variables

bash

```
nano ~/.bashrc

# Add the following lines:
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"

# Reload
source ~/.bashrc
```

Step 6: Configure Hadoop Files

Edit hadoop-env.sh:

bash

```
nano $HADOOP_HOME/etc/hadoop/hadoop-env.sh

# Add:
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

Edit core-site.xml:

bash

```
nano $HADOOP_HOME/etc/hadoop/core-site.xml
```

xml

```
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://localhost:9000</value>
    </property>
    <property>
        <name>hadoop.tmp.dir</name>
        <value>/home/hadoop/hadoopdata/tmp</value>
    </property>
</configuration>
```

Edit hdfs-site.xml:

bash

```
nano $HADOOP_HOME/etc/hadoop/hdfs-site.xml
```

xml

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>file:///home/hadoop/hadoopdata/namenode</value>
    </property>
    <property>
        <name>dfs.datanode.data.dir</name>
        <value>file:///home/hadoop/hadoopdata/datanode</value>
```

```xml
    </property>
</configuration>
```

Edit mapred-site.xml:

bash

```bash
nano $HADOOP_HOME/etc/hadoop/mapred-site.xml
```

xml

```xml
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
</configuration>
```

Edit yarn-site.xml:

bash

```bash
nano $HADOOP_HOME/etc/hadoop/yarn-site.xml
```

xml

```xml
<configuration>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
</configuration>
```

Step 7: Create Required Directories

bash

```bash
mkdir -p ~/hadoopdata/tmp
mkdir -p ~/hadoopdata/namenode
mkdir -p ~/hadoopdata/datanode
```

Step 8: Format NameNode and Start Hadoop

bash

```bash
# Format NameNode (only first time)
hdfs namenode -format

# Start Hadoop services
```

```
start-dfs.sh
start-yarn.sh

# Verify services
jps
```

Step 9: File Management Operations

Creating Directories:

bash

```
# Create directory in HDFS
hdfs dfs -mkdir /user
hdfs dfs -mkdir /user/hadoop
hdfs dfs -mkdir /input

# List directories
hdfs dfs -ls /
hdfs dfs -ls /user
```

Adding Files:

bash

```
# Create sample file
echo "Hello Hadoop! This is a test file." > sample.txt
echo "Line 1: Big Data Processing" > data.txt
echo "Line 2: Hadoop File System" >> data.txt

# Copy file from local to HDFS
hdfs dfs -put sample.txt /user/hadoop/
hdfs dfs -put data.txt /input/

# Copy entire directory
mkdir local_data
echo "File 1 content" > local_data/file1.txt
echo "File 2 content" > local_data/file2.txt
hdfs dfs -put local_data /user/hadoop/
```

Retrieving Files:

bash

```
# View file content
hdfs dfs -cat /user/hadoop/sample.txt

# Copy file from HDFS to local
hdfs dfs -get /user/hadoop/sample.txt downloaded_sample.txt

# Copy to current directory
hdfs dfs -get /input/data.txt .

# View file using copyToLocal
hdfs dfs -copyToLocal /user/hadoop/data.txt local_copy.txt
```

Deleting Files and Directories:

bash

```
# Delete a file
hdfs dfs -rm /user/hadoop/sample.txt

# Delete directory (non-empty)
hdfs dfs -rm -r /user/hadoop/local_data

# Delete directory (must be empty)
hdfs dfs -rmdir /test_dir

# Move to trash (recoverable)
hdfs dfs -rm /input/data.txt

# Permanent deletion
hdfs dfs -rm -skipTrash /input/data.txt
```

Additional Useful Commands:

bash

```
# Check file status
hdfs dfs -stat /user/hadoop/sample.txt

# Change replication factor
hdfs dfs -setrep -w 3 /user/hadoop/sample.txt

# Check disk usage
```

```
hdfs dfs -du /user/hadoop

# Check filesystem
hdfs dfsadmin -report
```

Experiment 2: Matrix Multiplication using MapReduce

Objective

Develop a MapReduce program to multiply two matrices.

Matrix Multiplication Logic

For matrices A (m×n) and B (n×p), Result C (m×p):

- C[i][j] = Σ(A[i][k] × B[k][j]) for k=0 to n-1

Java Implementation

MatrixMultiplication.java:

java

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

public class MatrixMultiplication {

  // Mapper Class
  public static class MatrixMapper extends Mapper<LongWritable, Text, Text, Text> {

    @Override
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {

      Configuration conf = context.getConfiguration();
      int m = Integer.parseInt(conf.get("m")); // rows in A
```

```
        int p = Integer.parseInt(conf.get("p")); // columns in B

        String line = value.toString();
        String[] parts = line.split(",");

        String matrix = parts[0]; // A or B
        int i = Integer.parseInt(parts[1]); // row
        int j = Integer.parseInt(parts[2]); // column
        int val = Integer.parseInt(parts[3]); // value

        if (matrix.equals("A")) {
            // For each element A[i][j], emit for all columns of B
            for (int k = 0; k < p; k++) {
                context.write(new Text(i + "," + k),
                            new Text("A," + j + "," + val));
            }
        } else { // matrix B
            // For each element B[j][k], emit for all rows of A
            for (int k = 0; k < m; k++) {
                context.write(new Text(k + "," + j),
                            new Text("B," + i + "," + val));
            }
        }
    }
}

// Reducer Class
public static class MatrixReducer extends Reducer<Text, Text, Text, Text> {

    @Override
    public void reduce(Text key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {

        Map<Integer, Integer> aElements = new HashMap<>();
        Map<Integer, Integer> bElements = new HashMap<>();

        // Separate A and B elements
        for (Text val : values) {
            String[] parts = val.toString().split(",");
            String matrix = parts[0];
            int index = Integer.parseInt(parts[1]);
            int value = Integer.parseInt(parts[2]);

            if (matrix.equals("A")) {
                aElements.put(index, value);
            } else {
                bElements.put(index, value);
            }
        }

        // Calculate dot product
        int sum = 0;
        for (Integer k : aElements.keySet()) {
            if (bElements.containsKey(k)) {
                sum += aElements.get(k) * bElements.get(k);
            }
        }

        context.write(key, new Text(String.valueOf(sum)));
    }
}

// Driver Code
public static void main(String[] args) throws Exception {

    Configuration conf = new Configuration();
    conf.set("m", "2"); // Set matrix dimensions
    conf.set("p", "2");

    Job job = Job.getInstance(conf, "Matrix Multiplication");
    job.setJarByClass(MatrixMultiplication.class);

    job.setMapperClass(MatrixMapper.class);
    job.setReducerClass(MatrixReducer.class);

    job.setOutputKeyClass(Text.class);
```

```
        job.setOutputValueClass(Text.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Steps to Execute

1. Create Input Files:

bash

```
# Create matrix_input.txt
# Format: Matrix,Row,Column,Value
cat > matrix_input.txt << EOF
A,0,0,1
A,0,1,2
A,1,0,3
A,1,1,4
B,0,0,5
B,0,1,6
B,1,0,7
B,1,1,8
EOF

# Upload to HDFS
hdfs dfs -mkdir /matrix
hdfs dfs -put matrix_input.txt /matrix/
```

2. Compile and Create JAR:

bash

```
# Set Hadoop classpath
export HADOOP_CLASSPATH=$(hadoop classpath)

# Compile
javac -classpath $HADOOP_CLASSPATH MatrixMultiplication.java
```

```
# Create JAR
jar -cvf matrixmult.jar *.class
```

3. Run MapReduce Job:

bash

```
hadoop jar matrixmult.jar MatrixMultiplication /matrix/matrix_input.txt /matrix/output

# View results
hdfs dfs -cat /matrix/output/part-r-00000
```

Python Implementation (using mrjob)

matrix_multiply.py:

python

```
from mrjob.job import MRJob
from mrjob.step import MRStep
import sys

class MatrixMultiply(MRJob):

    def configure_args(self):
        super(MatrixMultiply, self).configure_args()
        self.add_passthru_arg('--m', type=int, default=2, help='Rows in Matrix A')
        self.add_passthru_arg('--n', type=int, default=2, help='Cols in A, Rows in B')
        self.add_passthru_arg('--p', type=int, default=2, help='Columns in Matrix B')

    def steps(self):
        return [
            MRStep(mapper=self.mapper,
                   reducer=self.reducer)
        ]

    def mapper(self, _, line):
        parts = line.strip().split(',')
        matrix = parts[0]
        i = int(parts[1])
        j = int(parts[2])
        value = int(parts[3])
        if matrix == 'A':
            # Emit for each column in result matrix
            for k in range(self.options.p):
                yield (i, k), ('A', j, value)
        else:  # Matrix B
            # Emit for each row in result matrix
            for k in range(self.options.m):
                yield (k, j), ('B', i, value)

    def reducer(self, key, values):
        a_values = {}
        b_values = {}

        for value in values:
            matrix, index, val = value
            if matrix == 'A':
                a_values[index] = val
            else:
                b_values[index] = val

        result = sum(a_values.get(k, 0) * b_values.get(k, 0)
                     for k in range(self.options.n))

        yield key, result

if __name__ == '__main__':
    MatrixMultiply.run()
```

Run Python version:

bash

```
# Install mrjob
pip install mrjob

# Run locally
python matrix_multiply.py matrix_input.txt --m=2 --n=2 --p=2

# Run on Hadoop
```

```
python matrix_multiply.py -r hadoop hdfs:///matrix/matrix_input.txt --m
```

Experiment 3: Weather Data Mining with MapReduce

Objective

Analyze weather data and display appropriate messages indicating weather conditions.

Java Implementation

WeatherAnalyzer.java:

java

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import java.io.IOException;

public class WeatherAnalyzer {

  // Mapper Class
  public static class WeatherMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

    @Override
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {

      String line = value.toString();

      // Skip header
      if (line.startsWith("Date") || line.isEmpty()) {
        return;
      }

      String[] fields = line.split(",");
      try {
        String date = fields[0];
        String year = date.substring(0, 4);
        int temperature = Integer.parseInt(fields[1].trim());
        int humidity = Integer.parseInt(fields[2].trim());
        String condition = fields[3].trim();

        // Emit year with temperature
        context.write(new Text(year + "_TEMP"), new IntWritable(temperature));

        // Emit year with humidity
        context.write(new Text(year + "_HUMIDITY"), new IntWritable(humidity));

        // Emit condition count
        context.write(new Text(year + "_" + condition.toUpperCase()),
              new IntWritable(1));

      } catch (Exception e) {
        // Skip malformed lines
        System.err.println("Error parsing line: " + line);
      }
    }
  }

  // Reducer Class
  public static class WeatherReducer extends Reducer<Text, IntWritable, Text, Text> {

    @Override
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {

      int sum = 0;
      int count = 0;
      int max = Integer.MIN_VALUE;
      int min = Integer.MAX_VALUE;

      for (IntWritable val : values) {
```

```
    int value = val.get();
    sum += value;
    count++;
    if (value > max) max = value;
    if (value < min) min = value;
}

String keyStr = key.toString();
String message = "";

if (keyStr.contains("_TEMP")) {
    double avg = (double) sum / count;
    message = String.format("Avg: %.2f°C, Max: %d°C, Min: %d°C | ",
                          avg, max, min);

    if (avg > 35) {
        message += "WARNING: Very Hot Weather!";
    } else if (avg > 25) {
        message += "Warm Weather";
    } else if (avg > 15) {
        message += "Pleasant Weather";
    } else if (avg > 5) {
        message += "Cool Weather";
    } else {
        message += "Cold Weather";
    }

} else if (keyStr.contains("_HUMIDITY")) {
    double avg = (double) sum / count;
    message = String.format("Avg Humidity: %.2f%% | ", avg);

    if (avg > 80) {
        message += "Very Humid";
    } else if (avg > 60) {
        message += "Moderately Humid";
    } else {
        message += "Dry";
    }
```

```
            } else {
                // Weather condition counts
                message = "Total days: " + count;
            }

            context.write(key, new Text(message));
        }
    }

    // Driver Code
    public static void main(String[] args) throws Exception {

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Weather Analysis");

        job.setJarByClass(WeatherAnalyzer.class);
        job.setMapperClass(WeatherMapper.class);
        job.setReducerClass(WeatherReducer.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Sample Input Data

weather_data.csv:

csv

```
Date,Temperature,Humidity,Condition
2024-01-01,22,65,Sunny
2024-01-02,25,70,Cloudy
```

```
2024-01-03,18,80,Rainy
2024-01-04,28,60,Sunny
2024-01-05,32,55,Sunny
2024-01-06,15,85,Rainy
2024-01-07,20,75,Cloudy
2024-02-01,24,68,Sunny
2024-02-02,19,82,Rainy
2024-02-03,30,58,Sunny
```

Execution Steps

bash

```
# Create input file
hdfs dfs -mkdir /weather
hdfs dfs -put weather_data.csv /weather/

# Compile
export HADOOP_CLASSPATH=$(hadoop classpath)
javac -classpath $HADOOP_CLASSPATH WeatherAnalyzer.java
jar -cvf weather.jar *.class

# Run
hadoop jar weather.jar WeatherAnalyzer /weather/weather_data.csv /weather/output

# View results
hdfs dfs -cat /weather/output/part-r-00000
```

Python Implementation

weather_analyzer.py:

python

```
from mrjob.job import MRJob
from mrjob.step import MRStep

class WeatherAnalyzer(MRJob):

    def steps(self):
        return [
            MRStep(mapper=self.mapper_get_weather,
                   reducer=self.reducer_analyze_weather)
        ]

    def mapper_get_weather(self, _, line):
        if line.startswith('Date') or not line.strip():
            return

        try:
            fields = line.split(',')
            date = fields[0]
            year = date.split('-')[0]
            temp = int(fields[1])
            humidity = int(fields[2])
            condition = fields[3].strip()

            yield (year, 'TEMP'), temp
            yield (year, 'HUMIDITY'), humidity
            yield (year, condition.upper()), 1

        except:
            pass

    def reducer_analyze_weather(self, key, values):
        values_list = list(values)
        year, metric = key

        if metric == 'TEMP':
            avg_temp = sum(values_list) / len(values_list)
            max_temp = max(values_list)
            min_temp = min(values_list)

            message = f"Avg: {avg_temp:.2f}°C, Max: {max_temp}°C, Min: {min_temp}°C | "

            if avg_temp > 35:
                message += "WARNING: Very Hot Weather!"
            elif avg_temp > 25:
                message += "Warm Weather"
            elif avg_temp > 15:
                message += "Pleasant Weather"
            else:
                message += "Cool Weather"

            yield f"{year}_TEMPERATURE", message

        elif metric == 'HUMIDITY':
            avg_humidity = sum(values_list) / len(values_list)
            message = f"Avg Humidity: {avg_humidity:.2f}% | "

            if avg_humidity > 80:
                message += "Very Humid"
            elif avg_humidity > 60:
                message += "Moderately Humid"
            else:
                message += "Dry"

            yield f"{year}_HUMIDITY", message

        else:
            total = sum(values_list)
            yield f"{year}_{metric}", f"Total days: {total}"

if __name__ == '__main__':
    WeatherAnalyzer.run()
```

Run:

bash

```
python weather_analyzer.py weather_data.csv
```

Experiment 4: MovieLens Data Analysis - Finding Tags for Movies

Objective

Analyze MovieLens data to find tags associated with each movie.

Java Implementation

MovieTagAnalyzer.java:

java

```java
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.MultipleInputs;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Set;

public class MovieTagAnalyzer {

  // Mapper for movies.csv
  public static class MovieMapper extends Mapper<LongWritable, Text, Text, Text> {

    @Override
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {

      String line = value.toString();

      if (line.startsWith("movieId") || line.isEmpty()) {
        return;
      }
```

```
        try {
            // Parse: movieId,title,genres
            String[] parts = line.split(",(?=(?:[^\"]*\"[^\"]*\")*[^\"]*$)");

            if (parts.length >= 2) {
                String movieId = parts[0].trim();
                String title = parts[1].trim().replace("\"", "");

                // Emit: movieId -> M:title
                context.write(new Text(movieId), new Text("M:" + title));
            }
        } catch (Exception e) {
            System.err.println("Error in MovieMapper: " + e.getMessage());
        }
    }
}

// Mapper for tags.csv
public static class TagMapper extends Mapper<LongWritable, Text, Text, Text> {

    @Override
    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {

        String line = value.toString();

        if (line.startsWith("userId") || line.isEmpty()) {
            return;
        }

        try {
            // Parse: userId,movieId,tag,timestamp
            String[] parts = line.split(",");

            if (parts.length >= 3) {
                String movieId = parts[1].trim();
                String tag = parts[2].trim().replace("\"", "");
```

```
                    // Emit: movieId -> T:tag
                    context.write(new Text(movieId), new Text("T:" + tag));
                }
            } catch (Exception e) {
                System.err.println("Error in TagMapper: " + e.getMessage());
            }
        }
    }

    // Reducer to combine movies with their tags
    public static class MovieTagReducer extends Reducer<Text, Text, Text, Text> {

        @Override
        public void reduce(Text key, Iterable<Text> values, Context context)
                throws IOException, InterruptedException {

            String movieTitle = "Unknown";
            Set<String> tags = new HashSet<>();

            for (Text val : values) {
                String value = val.toString();

                if (value.startsWith("M:")) {
                    movieTitle = value.substring(2);
                } else if (value.startsWith("T:")) {
                    tags.add(value.substring(2));
                }
            }

            if (!tags.isEmpty()) {
                String tagList = String.join(", ", tags);
                String output = String.format("%s | Tags: %s (Total: %d)",
                                              movieTitle, tagList, tags.size());
                context.write(key, new Text(output));
            }
        }
    }
    // Driver Code
    public static void main(String[] args) throws Exception {

        if (args.length != 3) {
            System.err.println("Usage: MovieTagAnalyzer <movies.csv> <tags.csv> <output>");
            System.exit(-1);
        }

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Movie Tag Analysis");

        job.setJarByClass(MovieTagAnalyzer.class);
        job.setReducerClass(MovieTagReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        // Multiple inputs
        MultipleInputs.addInputPath(job, new Path(args[0]),
                                TextInputFormat.class, MovieMapper.class);
        MultipleInputs.addInputPath(job, new Path(args[1]),
                                TextInputFormat.class, TagMapper.class);

        FileOutputFormat.setOutputPath(job, new Path(args[2]));

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Sample Data Files

movies.csv:

csv

```
movieId,title,genres
1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy
2,Jumanji (1995),Adventure|Children|Fantasy
3,Grumpier Old Men (1995),Comedy|Romance
4,Waiting to Exhale (1995),Comedy|Drama|Romance
```

```
5,Father of the Bride Part II (1995),Comedy
```

tags.csv:

csv

```
userId,movieId,tag,timestamp
1,1,pixar,1234567890
2,1,fun,1234567891
3,1,animation,1234567892
4,2,adventure,1234567893
5,2,robin williams,1234567894
6,3,comedy,1234567895
7,1,classic,1234567896
```

Execution Steps

bash

```
# Upload data to HDFS
hdfs dfs -mkdir /movielens
hdfs dfs -put movies.csv /movielens/
hdfs dfs -put tags.csv /movielens/

# Compile
export HADOOP_CLASSPATH=$(hadoop classpath)
javac -classpath $HADOOP_CLASSPATH MovieTagAnalyzer.java
jar -cvf movietag.jar *.class

# Run
hadoop jar movietag.jar MovieTagAnalyzer /movielens/movies.csv /movielens/tags.csv /movielens/output

# View results
hdfs dfs -cat /movielens/output/part-r-00000
```

Python Implementation

movie_tag_analyzer.py:

python

```
from mrjob.job import MRJob
from mrjob.step import MRStep

class MovieTagAnalyzer(MRJob):
```

```
def steps(self):
    return [
        MRStep(mapper=self.mapper,
               reducer=self.reducer)
    ]

def mapper(self, _, line):
    if line.startswith('movieId') or line.startswith('userId') or not line.strip():
        return

    parts = line.split(',')

    # Check if it's movies.csv (has title field)
    if len(parts) >= 2 and not parts[0].isdigit():
        return

    # tags.csv format: userId,movieId,tag,timestamp
    if len(parts) >= 4:
        movie_id = parts[1].strip()
        tag = parts[2].strip().replace('"', '')
        yield movie_id, ('T', tag)

    # movies.csv format: movieId,title,genres
    elif len(parts) >= 2:
        movie_id = parts[0].strip()
        title = parts[1].strip().replace('"', '')
        yield movie_id, ('M', title)

def reducer(self, movie_id, values):
    movie_title = "Unknown"
    tags = set()

    for value_type, value in values:
        if value_type == 'M':
            movie_title = value
        elif value_type == 'T':
            tags.add(value)
        if tags:
            tag_list = ', '.join(sorted(tags))
            output = f"{movie_title} | Tags: {tag_list} (Total: {len(tags)})"
            yield movie_id, output

if __name__ == '__main__':
    MovieTagAnalyzer.run()
```

Run:

bash

```
python movie_tag_analyzer.py movies.csv tags.csv -o output
```

Experiment 5: MongoDB Operations - Count, Sort, Limit, Skip, Aggregate

Objective

Implement various MongoDB operations including count, sort, limit, skip, and aggregate functions.

Step 1: Install MongoDB

bash

```
# Import MongoDB public GPG key
wget -qO - https://www.mongodb.org/static/pgp/server-6.0.asc | sudo apt-key add -

# Create list file
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/6.0 multiverse" |
sudo tee /etc/apt/sources.list.d/mongodb-org-6.0.list

# Update and install
sudo apt update
sudo apt install -y mongodb-org

# Start MongoDB
sudo systemctl start mongod
sudo systemctl enable mongod

# Check status
sudo systemctl status mongod

# Access MongoDB shell
mongosh
```

Step 2: Create Sample Database and Collections

javascript

```
// Switch to database
use studentDB

// Insert sample student documents
db.students.insertMany([
  {
```

```
  name: "Alice Johnson",
  age: 20,
  grade: "A",
  marks: 95,
  city: "New York",
  courses: ["Math", "Physics", "Chemistry"]
},
{
  name: "Bob Smith",
  age: 22,
  grade: "B",
  marks: 82,
  city: "Los Angeles",
  courses: ["Math", "Computer Science"]
},
{
  name: "Charlie Brown",
  age: 21,
  grade: "A",
  marks: 88,
  city: "Chicago",
  courses: ["Physics", "Math", "Biology"]
},
{
  name: "Diana Prince",
  age: 23,
  grade: "C",
  marks: 75,
  city: "New York",
  courses: ["Chemistry", "Biology"]
},
{
  name: "Eve Davis",
  age: 20,
  grade: "A",
  marks: 92,
  city: "Boston",
  courses: ["Computer Science", "Math"]
  },
  {
    name: "Frank Miller",
    age: 24,
    grade: "B",
    marks: 78,
    city: "Chicago",
    courses: ["Physics", "Chemistry"]
  },
  {
    name: "Grace Lee",
    age: 22,
    grade: "A",
    marks: 90,
    city: "San Francisco",
    courses: ["Math", "Computer Science", "Physics"]
  },
  {
    name: "Henry Wilson",
    age: 21,
    grade: "B",
    marks: 85,
    city: "Seattle",
    courses: ["Biology", "Chemistry"]
  }
])
```

Operation 1: COUNT

javascript

```
// Count all documents
db.students.countDocuments()

// Count with filter
db.students.countDocuments({ grade: "A" })

// Count students from New York
db.students.countDocuments({ city: "New York" })

// Count students with marks > 85
```

```
db.students.countDocuments({ marks: { $gt: 85 } })

// Count students aged between 20 and 22
db.students.countDocuments({ age: { $gte: 20, $lte: 22 } })

// Count using estimated count (faster for large collections)
db.students.estimatedDocumentCount()
```

Operation 2: SORT

javascript

```
// Sort by marks (ascending)
db.students.find().sort({ marks: 1 })

// Sort by marks (descending)
db.students.find().sort({ marks: -1 })

// Sort by multiple fields
db.students.find().sort({ grade: 1, marks: -1 })

// Sort by name alphabetically
db.students.find().sort({ name: 1 })

// Sort and display only name and marks
db.students.find({}, { name: 1, marks: 1, _id: 0 }).sort({ marks: -1 })

// Sort by age (descending) and marks (descending)
db.students.find().sort({ age: -1, marks: -1 })
```

Operation 3: LIMIT

javascript

```
// Get first 3 students
db.students.find().limit(3)

// Get top 3 students by marks
db.students.find().sort({ marks: -1 }).limit(3)

// Get students with grade A, limit to 2
db.students.find({ grade: "A" }).limit(2)

// Get youngest 3 students
```

```
db.students.find().sort({ age: 1 }).limit(3)

// Combine with projection
db.students.find({}, { name: 1, marks: 1, _id: 0 })
        .sort({ marks: -1 })
        .limit(5)
```

Operation 4: SKIP

javascript

```
// Skip first 2 documents
db.students.find().skip(2)

// Skip first 3, get next 3 (pagination)
db.students.find().skip(3).limit(3)

// Sort, skip, and limit (pagination example)
db.students.find()
        .sort({ marks: -1 })
        .skip(2)
        .limit(3)

// Page 1 (first 3 records)
db.students.find().sort({ name: 1 }).limit(3)

// Page 2 (next 3 records)
db.students.find().sort({ name: 1 }).skip(3).limit(3)

// Page 3 (next 3 records)
db.students.find().sort({ name: 1 }).skip(6).limit(3)
```

Operation 5: AGGREGATE

Basic Aggregation:

javascript

```
// Average marks of all students
db.students.aggregate([
  {
    $group: {
      _id: null,
            avgMarks: { $avg: "$marks" },
            totalStudents: { $sum: 1 }
        }
    }
])

// Group by grade and count
db.students.aggregate([
    {
        $group: {
            _id: "$grade",
            count: { $sum: 1 },
            avgMarks: { $avg: "$marks" }
        }
    }
])

// Group by city
db.students.aggregate([
    {
        $group: {
            _id: "$city",
            studentCount: { $sum: 1 },
            avgAge: { $avg: "$age" },
            maxMarks: { $max: "$marks" },
            minMarks: { $min: "$marks" }
        }
    }
])
```

Complex Aggregation Pipeline:

javascript

```
// Multi-stage aggregation
db.students.aggregate([
    // Stage 1: Match students with marks > 80
    {
        $match: { marks: { $gt: 80 } }
    },
    // Stage 2: Group by grade
    {
      $group: {
        _id: "$grade",
        count: { $sum: 1 },
        avgMarks: { $avg: "$marks" },
        students: { $push: "$name" }
      }
    },
    // Stage 3: Sort by average marks
    {
      $sort: { avgMarks: -1 }
    }
])

// Aggregation with projection
db.students.aggregate([
    {
      $project: {
        name: 1,
        marks: 1,
        grade: 1,
        passed: { $gte: ["$marks", 75] }
      }
    }
])

// Unwind array and group
db.students.aggregate([
    { $unwind: "$courses" },
    {
      $group: {
        _id: "$courses",
        studentCount: { $sum: 1 },
        students: { $push: "$name" }
      }
    },
    { $sort: { studentCount: -1 } }
```

```
])

// Complex pipeline with multiple operations
db.students.aggregate([
    // Match grade A or B
    { $match: { grade: { $in: ["A", "B"] } } },
    // Add computed field
    {
        $addFields: {
            marksCategory: {
                $cond: {
                    if: { $gte: ["$marks", 90] },
                    then: "Excellent",
                    else: "Good"
                }
            }
        }
    },
    // Group by city
    {
        $group: {
            _id: "$city",
            avgMarks: { $avg: "$marks" },
            topStudent: { $max: "$marks" },
            studentList: { $push: { name: "$name", marks: "$marks" } }
        }
    },
    // Sort by average marks
    { $sort: { avgMarks: -1 } },
    // Limit to top 3 cities
    { $limit: 3 }
])
```

Statistical Aggregations:

javascript

```
// Calculate statistics
db.students.aggregate([
    {
```

```
    $group: {
      _id: null,
      totalStudents: { $sum: 1 },
      avgMarks: { $avg: "$marks" },
      maxMarks: { $max: "$marks" },
      minMarks: { $min: "$marks" },
      totalMarks: { $sum: "$marks" },
      stdDev: { $stdDevPop: "$marks" }
    }
  }
])

// Bucket aggregation (group by mark ranges)
db.students.aggregate([
  {
    $bucket: {
      groupBy: "$marks",
      boundaries: [0, 70, 80, 90, 100],
      default: "Other",
      output: {
        count: { $sum: 1 },
        students: { $push: "$name" }
      }
    }
  }
])
```

Complete Python Script for MongoDB Operations

mongodb_operations.py:

python

```
from pymongo import MongoClient
from pprint import pprint

# Connect to MongoDB
client = MongoClient('mongodb://localhost:27017/')
db = client['studentDB']
collection = db['students']
```

```
print("="*60)
print("MONGODB OPERATIONS DEMONSTRATION")
print("="*60)

# COUNT Operations
print("\n1. COUNT OPERATIONS:")
print("-" * 40)
print(f"Total students: {collection.count_documents({})}")
print(f"Students with grade A: {collection.count_documents({'grade': 'A'})}")
print(f"Students with marks > 85: {collection.count_documents({'marks': {'$gt': 85}})}")

# SORT Operations
print("\n2. SORT OPERATIONS:")
print("-" * 40)
print("Top 3 students by marks:")
for doc in collection.find({}, {'name': 1, 'marks': 1, '_id': 0}).sort('marks', -1).limit(3):
    print(f"  {doc['name']}: {doc['marks']}")

# LIMIT Operations
print("\n3. LIMIT OPERATIONS:")
print("-" * 40)
print("First 3 students:")
for doc in collection.find({}, {'name': 1, '_id': 0}).limit(3):
    print(f"  {doc['name']}")

# SKIP Operations
print("\n4. SKIP OPERATIONS (Pagination):")
print("-" * 40)
page_size = 3
page_num = 2
print(f"Page {page_num} (records {(page_num-1)*page_size + 1} to {page_num*page_size}):")
for doc in collection.find({}, {'name': 1, '_id': 0}).skip((page_num-1)*page_size).limit(page_size):
    print(f"  {doc['name']}")

# AGGREGATE Operations
print("\n5. AGGREGATE OPERATIONS:")
print("-" * 40)
```

```
# Average marks by grade
print("\nAverage marks by grade:")
pipeline = [
    {
        '$group': {
            '_id': '$grade',
            'avgMarks': {'$avg': '$marks'},
            'count': {'$sum': 1}
        }
    },
    {'$sort': {'avgMarks': -1}}
]
for doc in collection.aggregate(pipeline):
    print(f"  Grade {doc['_id']}: Avg={doc['avgMarks']:.2f}, Count={doc['count']}")

# Students per city
print("\nStudents per city:")
pipeline = [
    {
        '$group': {
            '_id': '$city',
            'count': {'$sum': 1},
            'avgMarks': {'$avg': '$marks'}
        }
    },
    {'$sort': {'count': -1}}
]
for doc in collection.aggregate(pipeline):
    print(f"  {doc['_id']}: {doc['count']} students, Avg marks={doc['avgMarks']:.2f}")

# Course popularity
print("\nCourse popularity:")
pipeline = [
    {'$unwind': '$courses'},
    {
        '$group': {
            '_id': '$courses',
            'studentCount': {'$sum': 1}
        }
    },
    {'$sort': {'studentCount': -1}}
]
for doc in collection.aggregate(pipeline):
    print(f"  {doc['_id']}: {doc['studentCount']} students")

print("\n" + "="*60)
```

Run Python script:

bash

```
pip install pymongo
python mongodb_operations.py
```

Experiment 6: Pig Latin Scripts

Objective

Use Pig Latin to perform sort, group, join, project, and filter operations on data.

Step 1: Install Pig

bash

```
# Download Pig
wget https://downloads.apache.org/pig/pig-0.17.0/pig-0.17.0.tar.gz

# Extract
tar -xzf pig-0.17.0.tar.gz
sudo mv pig-0.17.0 /usr/local/pig

# Set environment variables
echo "export PIG_HOME=/usr/local/pig" >> ~/.bashrc
echo "export PATH=\$PATH:\$PIG_HOME/bin" >> ~/.bashrc
echo "export PIG_CLASSPATH=\$HADOOP_HOME/etc/hadoop" >> ~/.bashrc
source ~/.bashrc

# Verify installation
pig -version
```

Step 2: Create Sample Data Files

students.txt:

```
1,Alice,85,Math,New York
2,Bob,78,Physics,Boston
3,Charlie,92,Math,Chicago
4,Diana,88,Chemistry,New York
5,Eve,75,Physics,Boston
6,Frank,95,Math,Seattle
7,Grace,82,Chemistry,Chicago
```

departments.txt:

```
Math,Science Building,Dr. Smith
Physics,Science Building,Dr. Johnson
Chemistry,Lab Building,Dr. Williams
```

```
Biology,Science Building,Dr. Brown
```

Upload to HDFS:

bash

```
hdfs dfs -mkdir /pig_data
hdfs dfs -put students.txt /pig_data/
hdfs dfs -put departments.txt /pig_data/
```

Operation 1: LOAD and PROJECT

project_demo.pig:

pig

```
-- Load data
students = LOAD '/pig_data/students.txt' USING PigStorage(',')
   AS (id:int, name:chararray, marks:int, subject:chararray, city:chararray);

-- Project specific columns
projected = FOREACH students GENERATE name, marks, subject;

-- Display
DUMP projected;

-- Store result
STORE projected INTO '/pig_data/output/projected' USING PigStorage(',');
```

Operation 2: FILTER

filter_demo.pig:

pig

```
-- Load data
students = LOAD '/pig_data/students.txt' USING PigStorage(',')
   AS (id:int, name:chararray, marks:int, subject:chararray, city:chararray);

-- Filter students with marks > 80
high_scorers = FILTER students BY marks > 80;

-- Filter by subject
math_students = FILTER students BY subject == 'Math';
```

```
-- Filter by multiple conditions
ny_high_scorers = FILTER students BY marks > 80 AND city == 'New York';

-- Display
DUMP high_scorers;
DUMP math_students;
DUMP ny_high_scorers;

-- Store
STORE high_scorers INTO '/pig_data/output/high_scorers' USING PigStorage(',');
```

Operation 3: SORT

sort_demo.pig:

pig

```
-- Load data
students = LOAD '/pig_data/students.txt' USING PigStorage(',')
   AS (id:int, name:chararray, marks:int, subject:chararray, city:chararray);

-- Sort by marks (ascending)
sorted_asc = ORDER students BY marks ASC;

-- Sort by marks (descending)
sorted_desc = ORDER students BY marks DESC;

-- Sort by multiple fields
multi_sort = ORDER students BY subject ASC, marks DESC;

-- Sort and limit
top_3 = LIMIT sorted_desc 3;

-- Display
DUMP sorted_desc;
DUMP top_3;

-- Store
STORE sorted_desc INTO '/pig_data/output/sorted_students' USING PigStorage(',');
```

Operation 4: GROUP

group_demo.pig:

pig

```
-- Load data
students = LOAD '/pig_data/students.txt' USING PigStorage(',')
    AS (id:int, name:chararray, marks:int, subject:chararray, city:chararray);

-- Group by subject
by_subject = GROUP students BY subject;

-- Count students per subject
subject_count = FOREACH by_subject GENERATE
    group AS subject,
    COUNT(students) AS student_count;

-- Group by city and calculate statistics
by_city = GROUP students BY city;
city_stats = FOREACH by_city GENERATE
    group AS city,
    COUNT(students) AS total_students,
    AVG(students.marks) AS avg_marks,
    MAX(students.marks) AS max_marks,
    MIN(students.marks) AS min_marks;

-- Display
DUMP subject_count;
DUMP city_stats;

-- Store
STORE subject_count INTO '/pig_data/output/subject_count' USING PigStorage(',');
STORE city_stats INTO '/pig_data/output/city_stats' USING PigStorage(',');
```

Operation 5: JOIN

join_demo.pig:

pig

```
-- Load students
```

```
students = LOAD '/pig_data/students.txt' USING PigStorage(',')
    AS (id:int, name:chararray, marks:int, subject:chararray, city:chararray);

-- Load departments
departments = LOAD '/pig_data/departments.txt' USING PigStorage(',')
    AS (dept_name:chararray, building:chararray, hod:chararray);

-- Inner Join
joined_data = JOIN students BY subject, departments BY dept_name;

-- Project relevant columns
result = FOREACH joined_data GENERATE
    students::name AS student_name,
    students::marks AS marks,
    students::subject AS subject,
    departments::building AS building,
    departments::hod AS hod;

-- Left Outer Join
left_join = JOIN students BY subject LEFT OUTER, departments BY dept_name;

-- Display
DUMP result;

-- Store
STORE result INTO '/pig_data/output/joined_data' USING PigStorage(',');
```

Complete Pig Script with All Operations

complete_analysis.pig:

pig

```
-- ====================================
-- COMPLETE PIG LATIN DEMONSTRATION
-- ====================================

-- 1. LOAD DATA
students = LOAD '/pig_data/students.txt' USING PigStorage(',')
    AS (id:int, name:chararray, marks:int, subject:chararray, city:chararray);
```

```
departments = LOAD '/pig_data/departments.txt' USING PigStorage(',')
    AS (dept_name:chararray, building:chararray, hod:chararray);

-- 2. PROJECT (Select specific columns)
projected = FOREACH students GENERATE name, marks, subject;

-- 3. FILTER (High scorers)
high_scorers = FILTER students BY marks >= 85;

-- 4. SORT (By marks descending)
sorted_students = ORDER students BY marks DESC;

-- 5. GROUP and AGGREGATE
-- Group by subject
by_subject = GROUP students BY subject;
subject_stats = FOREACH by_subject GENERATE
    group AS subject,
    COUNT(students) AS num_students,
    AVG(students.marks) AS avg_marks,
    MAX(students.marks) AS max_marks,
    MIN(students.marks) AS min_marks;

-- Group by city
by_city = GROUP students BY city;
city_summary = FOREACH by_city GENERATE
    group AS city,
    COUNT(students) AS total_students,
    AVG(students.marks) AS average_marks;

-- 6. JOIN
student_dept = JOIN students BY subject, departments BY dept_name;
joined_result = FOREACH student_dept GENERATE
    students::name AS student,
    students::marks AS marks,
    students::subject AS subject,
    departments::building AS building,
    departments::hod AS hod;
```

```
-- 7. DISTINCT
unique_subjects = DISTINCT (FOREACH students GENERATE subject);
unique_cities = DISTINCT (FOREACH students GENERATE city);

-- 8. LIMIT
top_5 = LIMIT sorted_students 5;

-- 9. UNION (if you have multiple datasets)
-- high_performers = UNION high_scorers, another_set;

-- DISPLAY RESULTS
DESCRIBE students;
DUMP high_scorers;
DUMP subject_stats;
DUMP city_summary;
DUMP joined_result;
DUMP top_5;

-- STORE RESULTS
STORE high_scorers INTO '/pig_data/output/high_scorers' USING PigStorage(',');
STORE subject_stats INTO '/pig_data/output/subject_stats' USING PigStorage(',');
STORE city_summary INTO '/pig_data/output/city_summary' USING PigStorage(',');
STORE joined_result INTO '/pig_data/output/student_departments' USING PigStorage(',');
STORE top_5 INTO '/pig_data/output/top_students' USING PigStorage(',');
```

Execute Pig Scripts

bash

```
# Run in local mode
pig -x local project_demo.pig

# Run in MapReduce mode
pig -x mapreduce complete_analysis.pig

# Run in Grunt shell (interactive)
pig
grunt> -- Type your Pig Latin commands here
grunt> quit

# Run with parameters
```

```
pig -param input=/pig_data/students.txt -param output=/pig_data/result script.pig

# Check output
hdfs dfs -cat /pig_data/output/high_scorers/part-r-00000
```

Advanced Pig Operations

advanced_pig.pig:

pig

```
-- Load data
students = LOAD '/pig_data/students.txt' USING PigStorage(',')
    AS (id:int, name:chararray, marks:int, subject:chararray, city:chararray);

-- COGROUP (Similar to full outer join)
by_subject_group = GROUP students BY subject;
by_city_group = GROUP students BY city;
cogrouped = COGROUP by_subject_group BY group, by_city_group BY group;

-- CROSS (Cartesian product - use cautiously)
students_sample1 = LIMIT students 2;
students_sample2 = LIMIT students 2;
crossed = CROSS students_sample1, students_sample2;

-- SPLIT (Split into multiple relations)
SPLIT students INTO
    excellent IF marks >= 90,
    good IF marks >= 75 AND marks < 90,
    average IF marks < 75;

-- SAMPLE (Random sampling)
sampled = SAMPLE students 0.5;  -- 50% sample

-- NESTED FOREACH
grouped = GROUP students BY subject;
nested_result = FOREACH grouped {
    sorted = ORDER students BY marks DESC;
    top_2 = LIMIT sorted 2;
    GENERATE group, top_2;
}
```

```
-- RANK
ranked = RANK students BY marks DESC;

-- Display
DUMP excellent;
DUMP good;
DUMP nested_result;
DUMP ranked;
```

Experiment 7: Hive - Create, Alter, and Drop Databases, Tables, Views, Functions, and Indexes

Prerequisites:

- Hadoop installed and running
- Hive installed and configured

Step-by-Step Implementation:

Step 1: Start Hadoop Services

bash

```
start-dfs.sh

start-yarn.sh
```

Step 2: Start Hive

bash

```
hive
```

A. DATABASE OPERATIONS

Step 3: Create Database

sql

```
CREATE DATABASE IF NOT EXISTS company_db;
```

Step 4: Show All Databases

sql

```
SHOW DATABASES;
```

Step 5: Use Database

sql

```
USE company_db;
```

Step 6: Alter Database (Add Properties)

sql

```
ALTER DATABASE company_db SET DBPROPERTIES ('created_by' = 'admin', 'date' = '2025-10-30');
```

Step 7: Describe Database

sql

```
DESCRIBE DATABASE EXTENDED company_db;
```

B. TABLE OPERATIONS

Step 8: Create Table

sql

~~~
CREATE TABLE IF NOT EXISTS employees (
    emp_id INT,
    emp_name STRING,
    department STRING,
    salary DOUBLE,
    join_date DATE
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
~~~

Step 9: Show Tables

sql

~~~
SHOW TABLES;
~~~

Step 10: Describe Table

sql

~~~
DESCRIBE employees;
DESCRIBE FORMATTED employees;
```
~~~

#### **Step 11: Load Data into Table**

Create a sample data file (employees.csv):

~~~
```
1,John Doe,IT,75000,2020-01-15
2,Jane Smith,HR,65000,2019-03-22
3,Bob Johnson,Finance,80000,2021-06-10
4,Alice Brown,IT,72000,2020-09-05
5,Charlie Wilson,Marketing,68000,2021-02-18
~~~

Load data:

sql

~~~
LOAD DATA LOCAL INPATH '/path/to/employees.csv' INTO TABLE employees;
~~~

Step 12: Query Data

sql

```
SELECT * FROM employees;
SELECT * FROM employees WHERE department = 'IT';
```

Step 13: Alter Table (Add Column)

sql

```
ALTER TABLE employees ADD COLUMNS (email STRING);
```

Step 14: Alter Table (Rename Column)

sql

```
ALTER TABLE employees CHANGE emp_name employee_name STRING;
```

Step 15: Alter Table (Rename Table)

sql

```
ALTER TABLE employees RENAME TO staff;
```

C. VIEW OPERATIONS

Step 16: Create View

sql

```
CREATE VIEW IF NOT EXISTS it_employees AS
SELECT emp_id, employee_name, salary
FROM staff
WHERE department = 'IT';
```

Step 17: Show Views

sql

```
SHOW VIEWS;
```

Step 18: Query View

sql

```
SELECT * FROM it_employees;
```

Step 19: Drop View

sql

```
DROP VIEW IF EXISTS it_employees;
```

D. INDEX OPERATIONS (Note: Deprecated in Hive 3.0+)

Step 20: Create Index (If supported)

sql

```
CREATE INDEX emp_idx ON TABLE staff(emp_id)
AS 'COMPACT'
WITH DEFERRED REBUILD;
```

Step 21: Show Indexes

sql

```
SHOW INDEXES ON staff;
```

Step 22: Drop Index

sql

```
DROP INDEX IF EXISTS emp_idx ON staff;
```

E. FUNCTION OPERATIONS

Step 23: Show Built-in Functions

sql

```
SHOW FUNCTIONS;
```

Step 24: Describe Function

sql

```
DESCRIBE FUNCTION upper;
DESCRIBE FUNCTION EXTENDED upper;
```

Step 25: Create Temporary Function (Example with UDF)

First, you need a Java UDF class compiled into a JAR file.

sql

```
ADD JAR /path/to/your/udf.jar;
CREATE TEMPORARY FUNCTION my_upper AS 'com.example.MyUpperUDF';
```

Step 26: Use Custom Function

sql

```
SELECT my_upper(employee_name) FROM staff;
```

Step 27: Drop Function

sql

```
DROP TEMPORARY FUNCTION IF EXISTS my_upper;
```

F. CLEANUP OPERATIONS

Step 28: Drop Table

sql

```
DROP TABLE IF EXISTS staff;
```

Step 29: Drop Database

sql

```
DROP DATABASE IF EXISTS company_db CASCADE;
```

Step 30: Exit Hive

sql

```
exit;
```

Experiment 8: Word Count in Hadoop and Spark

PART A: Word Count in Hadoop MapReduce

Step 1: Create Input Data

bash

```
mkdir -p ~/hadoop_wordcount
cd ~/hadoop_wordcount
```

Create a file `input.txt`:

bash

```
cat > input.txt << EOF
Hello World
Hello Hadoop
Hadoop MapReduce
MapReduce Word Count
Word Count Example
EOF
```

Step 2: Start Hadoop Services

bash

```
start-dfs.sh
start-yarn.sh
```

Step 3: Create HDFS Directory and Upload Input

bash

```
hdfs dfs -mkdir -p /user/$(whoami)/wordcount/input
hdfs dfs -put input.txt /user/$(whoami)/wordcount/input/
```

Step 4: Verify Upload

bash

```
hdfs dfs -ls /user/$(whoami)/wordcount/input/
hdfs dfs -cat /user/$(whoami)/wordcount/input/input.txt
```

Step 5: Create WordCount Java Programme

Create file: WordCount.java

java

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

  public static class TokenizerMapper
      extends Mapper<Object, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context)
        throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
      extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
```

```
            }
            result.set(sum);
            context.write(key, result);
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Step 6: Compile the Java Programme

bash

```
export HADOOP_CLASSPATH=$(hadoop classpath)
javac -classpath $HADOOP_CLASSPATH -d . WordCount.java
```

Step 7: Create JAR File

bash

```
jar -cvf wordcount.jar *.class
```

Step 8: Run MapReduce Job

bash

```
hadoop jar wordcount.jar WordCount /user/$(whoami)/wordcount/input /user/$(whoami)/wordcount/output
```

Step 9: View Output

bash

```
hdfs dfs -ls /user/$(whoami)/wordcount/output/
hdfs dfs -cat /user/$(whoami)/wordcount/output/part-r-00000
```

```
```

Expected Output:

```
Count      2
Example    1
Hadoop     2
Hello      2
MapReduce            2
Word       2
World      1
```

PART B: Word Count in Spark

Step 1: Start Spark Shell (Python)

bash

```
pyspark
```

Step 2: Create RDD from Input File

python

```
# Read file from HDFS
text_file = sc.textFile("hdfs:///user/$(whoami)/wordcount/input/input.txt")
```

Step 3: Implement Word Count

python

```
# Word count transformation
counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)

# Collect and print results
output = counts.collect()
for (word, count) in output:
    print(f"{word}: {count}")
```

Step 4: Save Output to HDFS

python

```
counts.saveAsTextFile("hdfs:///user/$(whoami)/wordcount/spark_output")
```

Step 5: View Spark Output

python

```
exit()
```

bash

```
hdfs dfs -cat /user/$(whoami)/wordcount/spark_output/part-00000
```

Alternative: Spark Word Count using Scala

Step 1: Start Spark Shell (Scala)

bash

```
spark-shell
```

Step 2: Execute Word Count

scala

```
val textFile = sc.textFile("hdfs:///user/$(whoami)/wordcount/input/input.txt")
val counts = textFile.flatMap(line => line.split(" "))
                .map(word => (word, 1))
                .reduceByKey(_ + _)
counts.collect().foreach(println)
counts.saveAsTextFile("hdfs:///user/$(whoami)/wordcount/spark_scala_output")
```

Step 3: Exit

scala

```
:quit
```

Alternative: Spark Submit (Standalone Python Script)

Create file: spark_wordcount.py

python

```
from pyspark import SparkContext, SparkConf

conf = SparkConf().setAppName("Word Count")
sc = SparkContext(conf=conf)

text_file = sc.textFile("hdfs:///user/$(whoami)/wordcount/input/input.txt")
counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)

counts.saveAsTextFile("hdfs:///user/$(whoami)/wordcount/spark_submit_output")
sc.stop()
```

Run using spark-submit:

bash

```
spark-submit spark_wordcount.py
```

Experiment 9: Use CDH and HUE to Analyze Data and Generate Reports

Prerequisites:

- Cloudera Distribution Hadoop (CDH) installed
- HUE (Hadoop User Experience) configured

Step-by-Step Implementation:

Step 1: Start CDH Services

bash

```
# Start Cloudera Manager
sudo systemctl start cloudera-scm-server

# Or start services individually
sudo service hadoop-hdfs-namenode start
sudo service hadoop-hdfs-datanode start
sudo service hadoop-yarn-resourcemanager start
sudo service hadoop-yarn-nodemanager start
sudo service hive-server2 start
sudo service hue start
```

#### **Step 2: Access HUE Web Interface**

- Open browser and navigate to: `http://localhost:8888`
- Login with your credentials (default: admin/admin for new installations)

### **Step 3: Upload Sample Dataset**

#### **Via HUE File Browser:**

1. Click on **"File Browser"** in top menu
2. Navigate to `/user/your_username/`
3. Click **"Upload"** button
4. Select your sample dataset (CSV file)

Sample Dataset (sales_data.csv):

```
order_id,product,category,quantity,price,order_date
1,Laptop,Electronics,2,45000,2024-01-15
2,Mouse,Electronics,5,500,2024-01-16
3,Desk,Furniture,1,15000,2024-01-17
4,Chair,Furniture,4,8000,2024-01-18
5,Monitor,Electronics,3,12000,2024-01-19
6,Keyboard,Electronics,6,1500,2024-01-20
7,Table,Furniture,2,20000,2024-01-21
8,Headphones,Electronics,10,2000,2024-01-22
```

Or Upload via Command Line:

bash

```
hdfs dfs -mkdir -p /user/$(whoami)/sales_data
hdfs dfs -put sales_data.csv /user/$(whoami)/sales_data/
```

Step 4: Create Hive Table via HUE

1. Click on **"Query"** → **"Editor"** → **"Hive"**
2. Enter the following DDL:

sql

```
CREATE EXTERNAL TABLE IF NOT EXISTS sales (
    order_id INT,
    product STRING,
    category STRING,
    quantity INT,
    price DOUBLE,
    order_date DATE
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/user/your_username/sales_data'
TBLPROPERTIES ("skip.header.line.count"="1");
```

3. Click **"Execute"** button (or press Ctrl+Enter)

Step 5: Query Data via HUE

Query 1: View All Data

sql

```
SELECT * FROM sales LIMIT 10;
```

Query 2: Sales by Category

sql

```
SELECT
    category,
    COUNT(*) as total_orders,
    SUM(quantity) as total_quantity,
    SUM(quantity * price) as total_revenue
FROM sales
GROUP BY category
ORDER BY total_revenue DESC;
```

Query 3: Top Products by Revenue

sql

```
SELECT
    product,
    SUM(quantity * price) as revenue
FROM sales
GROUP BY product
ORDER BY revenue DESC
LIMIT 5;
```

Query 4: Daily Sales Analysis

sql

```
SELECT
    order_date,
    COUNT(*) as orders,
    SUM(quantity * price) as daily_revenue
FROM sales
GROUP BY order_date
ORDER BY order_date;
```

Step 6: Create Visualizations in HUE

1. After executing a query, click on the **chart icon** in the results section

2. Select chart type:
    - **Bar Chart**: For category comparisons
    - **Pie Chart**: For category distribution
    - **Line Chart**: For time series analysis
    - **Table**: For detailed data view

Example - Create Bar Chart:

- X-Axis: category
- Y-Axis: total_revenue
- Chart Type: Bar Chart
- Click **"Save"** to save the visualization

Step 7: Create Dashboard

1. Click on **"Dashboard"** in top menu
2. Click **"Create Dashboard"**
3. Add widgets:
    - Click **"+"** button
    - Select **"Add Query"**
    - Choose saved queries
    - Arrange widgets by dragging
4. Add filters (optional):
    - Click **"Add Filter"**
    - Select column (e.g., category, order_date)
5. Save Dashboard with a name

Step 8: Generate Reports

Method 1: Export Query Results

1. Execute your analysis query
2. Click **"Download"** icon in results section
3. Choose format: CSV, Excel, or JSON

Method 2: Schedule Reports

1. After executing query, click **"Schedule"** button
2. Set schedule:
    - Frequency: Daily/Weekly/Monthly
    - Time: Select time
    - Email recipients
3. Click **"Submit"**

Method 3: Create PDF Report

1. From Dashboard view
2. Click **"Export"** → **"PDF"**
3. Report will be downloaded

Step 9: Use Impala for Faster Queries (Optional)

1. Switch to **Impala Editor** in HUE
2. Refresh metadata:

sql

```
INVALIDATE METADATA sales;
```

3. Run queries (same syntax as Hive but faster):

sql

```
SELECT category, SUM(quantity * price) as revenue
FROM sales
GROUP BY category;
```

Step 10: Monitor Jobs

1. Click on **"Job Browser"** in top menu
2. View running/completed jobs:
   - MapReduce jobs
   - Spark jobs
   - Hive queries
3. Click on job to see:
   - Progress
   - Logs
   - Performance metrics

Step 11: Use Pig Editor (Optional)

1. Click on **"Query"** → **"Editor"** → **"Pig"**
2. Write Pig script:

pig

```
sales_data = LOAD '/user/your_username/sales_data/sales_data.csv'
   USING PigStorage(',')
```

```
    AS (order_id:int, product:chararray, category:chararray,
        quantity:int, price:double, order_date:chararray);

grouped = GROUP sales_data BY category;

result = FOREACH grouped GENERATE
    group as category,
    COUNT(sales_data) as total_orders,
    SUM(sales_data.quantity * sales_data.price) as revenue;

DUMP result;
```

3. Execute and view results

Step 12: Access HDFS Browser

1. Click on **"File Browser"**
2. Navigate through HDFS directories
3. Perform operations:
   - View file contents
   - Download files
   - Upload files
   - Delete files
   - Change permissions

Step 13: Create Sample Report Summary

Final Report Query:

sql

```
SELECT
    'Total Orders' as metric,
    CAST(COUNT(*) AS STRING) as value
FROM sales

UNION ALL

SELECT
    'Total Revenue',
    CAST(SUM(quantity * price) AS STRING)
FROM sales

UNION ALL

SELECT
    'Average Order Value',
    CAST(AVG(quantity * price) AS STRING)
FROM sales

UNION ALL

SELECT
    'Top Category',
    category
FROM (
    SELECT category, SUM(quantity * price) as revenue
    FROM sales
    GROUP BY category
    ORDER BY revenue DESC
    LIMIT 1
) t;
```

Cleanup (Optional)

bash

```
# Stop services
sudo service hue stop
sudo service hive-server2 stop

# Or stop all via Cloudera Manager
sudo systemctl stop cloudera-scm-server
```